In this case, users wait an average of only 122.5 milliseconds instead of 242.5, obtained using first-come, first served scheduling. More benefit is delivered sooner, and only less valuable and more costly jobs are delayed. Additionally, a pre-emptive scheduler has the ability, continuously, to insert higher priority jobs in front of lower priority ones. So, with the above scheduling and pre-emptive scheduling, the final, 200-millisecond job may run later than fourth, but it will not run sooner.

The invention can employs well-known techniques for optimization and job scheduling. With the present invention known optimization and job scheduling techniques can be used to provide efficient Web and Internet servers, independent of the particular optimization or scheduling technique used

Classification can be done as a mathematical function of known and estimated parameters. The above-mentioned benefit value may be a function of many parameters such as: requested URL, client IP address, cookie, login, connection quality and other distinguishing attributes. The cost may be more than just the time required to send or process the request, including, such factors as the required bandwidth, CPU, latency, data generation requirements, and total server load.

In general, the benefit and cost are functions of known and estimated parameters can be described as follows:

Benefit = B(p1, p2, p3...)
Cost = B(p1, p2, p3...), where pN are known and estimated parameters.

The functions may be tabular, or an actual mathematical function of the parameters or a combination of tabular and arithmetic functions. For example, a system can award points for a desirable URL request, one that is know to encourage commerce, compared to a reference or general information section of the site, which is read by both customers and non-customers.

Order placement = 100 points

Browse catalog = 20 points
Download reference material = 10 points

Points also go to requests that correspond to requestors who are, for example, good customers, as determined by the cookie, login, or, possibly, the IP source address.

Returning customer = +20 points

The connection quality determines how critical a connection is and how noticeable delays will be. This allows re-sequencing non-critical requests behind critical ones. In some cases, a modem user may not notice a slight delay but the DSL user will.

RTT estimate < 100 ms, critical, speed sensitive
RTT estimate > 100 ms , non-critical

The cost function is usually an estimate of the CPU required to generate the reply, the total time including latency required to generate the reply and the bandwidth required to send the reply back to the client. Other considerations include things such as the need for slots on application and database servers.

For example, a typical response may cost 2 ms CPU, 20 ms latency, and 25k Bytes of data transfer. In general the optimal scheduler is one that delivers the maximum benefit, subject to the constraints that the total costs are less than the system limits in all cases.

Example Two: Server connection and load management: A special case of a managed resource is a secondary server, usually an application server or database server. The server may suffer performance problems if its load is too high or if there are too many connections to the server from remote clients.

Typical server response time vs. load
1 pending requests -- 10 ms avg. response time
5 pending requests -- 12 ms avg. response time

10 pending requests – 20 ms avg. response time
20 pending requests -- 50 ms avg. response time
100 pending requests – 400 ms avg. response time

Here we see that, initially, efficiency increases due to increased concurrency and overlapping of requests that have both a latency (delay) and a processing (CPU) component. After a certain point, the server becomes less efficient due to overhead of maintaining many pending requests. Many application and database servers use operating system threads or processes to handle simultaneous tasks. This results in diminishing returns as threads corresponding to pending requests compete for synchronization primitives and as the operating system is forced to switch back and forth among the outstanding tasks.

In the above example, the server runs most efficiently at a load of around 10 pending requests, 20 ms average response time, for a total of around 500 "hits" per second. If the load is 100, with an average response time of 400 ms, then the throughput is about 250 hits per second. Intelligent load management would maintain the load on the server at 10, while queuing the remaining requests. As described in Example 1, this queuing has the added benefit of being able to order or prioritize the requests within the queue, with additional gains in throughput and reduction of average response time.

With the present invention requests can be handled by an intermediate server/optimizer, which queues the connection and transfers the data back and forth between the requesting client and the origin and application servers.

Net result due to intelligent load management, with 100 pending requests:
500 hits/sec, with an average response time of 20+ (2ms)*90 = 200 ms, compared with 250 hits/sec and 400 ms average response time with the standard application server.

Often, simply connecting to application and database servers slows the progress of tasks executing on the server. In this case, it is helpful to off-load idle connections to an intermediate server, which handles connection with an efficient queuing and I/O system.

A program flow diagram illustrating the operation of the system is given in Figure 2. First as indicated by block 201, the owner establishes a set of classifications priorities. These are stored in data base 123. If such a set are not as yet established the system utilizes a default set of priorities and classifications. The system receives requests from clients 100A to 100Z as indicated by block 203.

As indicated by block 295, the requests are classified in accordance with the classifications established by the system owner and stored in data base 123. Next as indicated by block 207, each request is prioritized in accordance with its classifications. Finally as indicated by block 208, the requests are scheduled in accordance with their priorities. Naturally higher priority requests are scheduled to be processes before lower priority requests.

Finally as indicated by block 209 the requests are sent to the web application server 110 and any outgoing traffic is sent to the network interface 120 for dispatch to the client machines 100A to 100Z.

Program 124 schedules the requests according to their priority and then prioritizes the requests in buffer 125. The requests are sent to the web application server 110 in accordance with their priority. A low priority requests which arrived first may reach web application server 112 after a high priority request which the system received at a later time.

In summary, the system includes programs that classify and priorities requests according to parameters established by the system operator. Different types of requests are provided with different priorities such that high priority requests are acted upon by the web application server 110 before low priority requests. This gives the system operator a great deal of flexibility in arranging the system provide a desired type of performance.

While in the embodiment described above, the classification, prioritization, and scheduling are done by three separate program routines, it should be understand, that the present invention relates to a program and system that performs this combination of functions. Those skilled in the art will realize that these functions can be performed using a wide variety of programming arrangements other than three separate programs.

In one embodiment of the present information, the classification, prioritization and scheduling is done based upon the TCP payload data that is received at the system 101. It is however noted that IP packets generally have the following form:
IP header gives source and destination information for routing.
TCP header: tells how may bytes in the payload, TCP options, etc.
TCP payload: Data bytes that contain a command such as that shown in Figure 3.

While one embodiment of the present invention operates by classifying, prioritizing and scheduling requests based upon the payload data, alternate embodiments take into account the information in the IP header and TCP header when the system does the classification, prioritzation and scheduling.

The above described embodiment of the invention merely improves performance by scheduling the order in which tasks are operated upon by the server. In an alternate embodiment, in addition to scheduling when the tasks witll be provided to the server, the amount of resources applied to each task is controlled in accordance with the classification and priority of the task. For example, the amount of memory that the server devotes to each class is controlled in accordance with the classification and priority of the particular task. In such an embodiment, when each request is passed to the server, a control parameter is also passed to the server. The control parameter instructs the server concerning the amount of resources (for example memory) that should be applied to the particular request.

The present invention optimizes CPU usage and network bandwidth while reducing latency and providing feedback to server administrators. This is accomplished by generating a cost-benefit model and optimizing it through request and task prioritization. The invention can be implemented as a custom kernel with a lean thread model further reduces requirements over a standard operating system's general-purpose thread model. Essentially, the custom scheduler and resource allocator take control away from the generic OS, returning it to the server owner. The custom kernel is fully event-driven, responding quickly to common conditions such as data ready or disk I/O complete, as well as to exception conditions like "connection reset by peer" or "address now unreachable". The kernel layer is designed to conserve system resources, especially RAM, so that the server will function optimally and degrade gracefully. Most systems exhibit non-linear delay vs. load characteristics, with a sharp knee in the curve at a critical load, indicating non-graceful degradation. The present invention will extend the curve by deferring lower priority and "housekeeping" tasks and by using system resources more efficiently.

The cost-benefit model enables prioritization of requests by content (URL) or requester (IP address, login, cookie), as well as according to more automatic criteria such as content length, resource requirements, or end-to-end connection quality. For example, if the server has one large request and ten small requests it may wish to service the ten small requests first, satisfying ten users, while adding an acceptable delay to the large request. Furthermore, shaving 100 milliseconds off a 200 millisecond RTT (round trip time) task would result in noticeable increase in interactivity. However, shaving 100 milliseconds off a 600-millisecond modem connection would not even be appreciated. This targeted, fine-grain optimization is enabled by characterizing the requests, estimating their resource requirements, then queuing up the required tasks in the correct sequence to optimize the model.

A server in accordance with the present invention improves efficiency by performing resource allocation and scheduling according to a cost-benefit model that is established both automatically and by the server's administrator. Requests to the server may be classified according to URL, URL parameters, requestor, connection quality, content size and generation requirements, server required, time of day, or any other identifying characteristic.

Each class of request may have its own priority level, benefit, maximum or weighted proportional share of total bandwidth, maximum or weighted proportional share of an assigned CPU, or priority of access to any system resource, server, or storage device.

Each class may have deadlines or constraints on delivery, with variable penalties for lateness and dropping. External, user objectives and constraints are translatable to internal constraints, which determine CPU and bandwidth scheduling and proportional share at the segment (packet) granularity.

With the present invention, a custom stack can schedules packet (segment) departure based on deadlines and lateness penalties that have been established by the scheduler and allocator

The invention utilizes an event-driven framework. A custom OS layer which manages the classification, prioritization and scheduling reduces the overhead of the general-purpose OS in the server, and it provides better communication between multiple, simultaneous tasks that are in progress. The custom OS layer maps multiple request/reply tasks to fewer threads or a single thread of the host OS.

The custom OS layer uses knowledge of continuations and non-blocking activities, co-operative multi-tasking based on a trust relationship between tasks is enabled. This is similar to the technique that is often used in the design of simulators. Such techniques reduce the overhead of multiprogramming a large number of independent tasks. Monitoring or a "pulse function" detects blocked or deadlocked processes to transfer workload to other, functioning processes, of which, new server OS instances may be added as needed.

A customized protocol stack can reduce the cost of open connections that have no assigned or discernible pending tasks. Such connections store only a source address and port without the usual socket resources. This stack may run in parallel with the existing TCP/IP stack by intercepting relevant ports, protocols, and URL requests at the kernel level, affording them special treatment.

The present invention eliminates network layer overhead. The custom stack also provides feedback to the scheduler and allocator regarding connection quality and available TCP and IP resources. Similarly, the custom stack greatly reduces the cost of servicing in-memory "cached" replies by forgoing the need for creating "socket" resources to grant access of kernel data to user spaces.

The custom stack may multiplex multiple connections and data transfers “on the wire” and at the network layer into a single user-level connection at the OS-user/application layer.

With the present invention, customized applications replace the layered network interface with interprocess communication (i.e. IPC) and remote procedure calls (i. e. RPC) to communicate directly with the system more efficiently.

One valuable by-product of peeking into the TCP layer to glean connection information is the ability to provide the server owner with more detailed traffic statistics. The server statistics and quality of achieving the user-defined and automatic criteria is fed-back to a monitoring and reporting application, which then displays said information to the administrator.

In conclusion: The present invention provides an event-driven custom kernel for a server. The custom kernel provides scheduling and resource allocation. The custom kernel operates in accordance with a cost-benefit model which is optimized by request and task prioritization. The cost-benefit model prioritizes requests by content (URL or requestor (IP address, login, or cookies) as well as according to criteria such as content length, resource requirements, or end-to-end connection quality. Tasks are classified and prioritized before being run on the CPU. Bandwidth is regulated and data departure is scheduled according to task and server specific criteria that can be established by a user. Fine-grain optimization is achieved by characterizing the requests, estimating their resource requirements, then queuing up the required tasks in the correct sequence to optimize the model. Several types of Interprocess communication (i.e. IPC) and remote procedure calls (i. e. RPC) are used to efficiently communicate directly with the system. These include providing feedback information between layers, and sending data directly from an internal layer to a receiver that is not an adjacent layer via inter-process communication. Since the kernel in the server obtains information from the TCP and IP layers, detailed traffic statistics can be provided to the server owner.

While the invention has been shown and descried with respect to preferred embodiments thereof, it will be understood by those skilled in the art, the various changes in form and detail can be made without departing from the spirit and scope of the present invention. The invention is limited only by the appended claims.

I claim: